JavaFX

Working with Layouts in JavaFX

Release 2.1

E20476-05

June 2013

Learn how to use the Layout API and built-in layout panes to lay out the interface for your JavaFX application.

ORACLE®

JavaFX Working with Layouts in JavaFX, Release 2.1

E20476-05

Primary Author: Joni Gordon

Contents

Part I

About This Tutorial

The JavaFX SDK provides layout panes that support several different styles of layouts. This tutorial provides information on using these panes to create graphical user interfaces for your JavaFX applications.

An intermediate level of Java language programming skills is assumed.

This tutorial contains the following topics:

- Using Built-in Layout Panes - Describes the built-in layout panes and provides an example of each pane.
- Tips for Sizing and Aligning Nodes - Provides examples of overriding the default size and position of nodes.
- Styling Layout Panes with CSS - Describes how to customize layout panes using CSS.

Expand the Table of Contents in the sidebar for a more detailed list of topics.

1

Using Built-in Layout Panes

This topic describes the layout container classes, called panes, that are available with the JavaFX SDK. Use layout panes to easily manage the user interface for your JavaFX application.

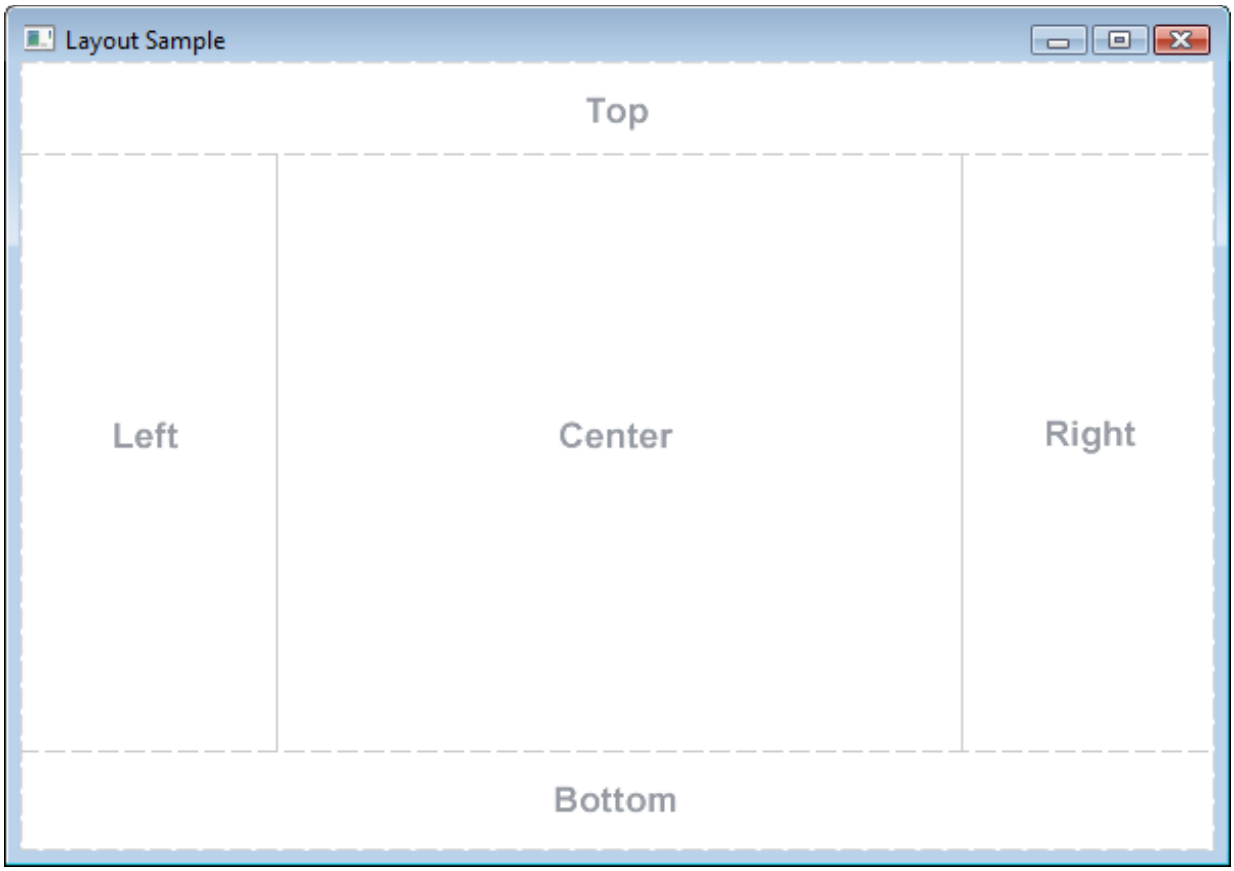
A JavaFX application can manually lay out the UI by setting the position and size properties for each UI element. However, an easier option is to make use of layout panes. The JavaFX SDK provides several layout panes for the easy setup and management of classic layouts such as rows, columns, stacks, tiles, and others. As a window is resized, the layout pane automatically repositions and resizes the nodes that it contains according to the properties for the nodes.

This topic provides an overview and a simple example of each of the layout panes provided by the JavaFX layout package. The LayoutSample.java file contains the source code for the UI built in this topic. The LayoutSample.zip file contains the NetBeans IDE project for the sample application.

BorderPane

The `BorderPane` layout pane provides five regions in which to place nodes: top, bottom, left, right, and center. Figure 1–1 shows the type of layout that you can create with a border pane. The regions can be any size. If your application does not need one of the regions, you do not need to define it and no space is allocated for it.

Figure 1–1 Sample Border Pane



A border pane is useful for the classic look of a tool bar at the top, a status bar at the bottom, a navigation panel on the left, additional information on the right, and a working area in the center.

If the window is larger than the space needed for the contents of each region, the extra space is given to the center region by default. If the window is smaller than the space needed for the contents of each region, the regions might overlap. The overlap is determined by the order in which the regions are set. For example, if the regions are set in the order of left, bottom, and right, when the window is made smaller, the bottom region overlaps the left region and the right region overlaps the bottom region. If set in the order of left, right, and bottom, when the window is made smaller, the bottom region overlaps both the left and right regions.

Example 1–1 shows the code for creating the border pane that is used for the UI that is built by the Layout Sample application. The methods that create the layout panes used in each region are described in the remaining sections of this topic.

Example 1–1 Create a Border Pane

```
BorderPane border = new BorderPane();
HBox hbox = addHBox()
border.setTop(hbox);
border.setLeft(addVBox());
addStackPane(hbox);         // Add stack to HBox in top region

border.setCenter(addGridPane());
border.setRight(addFlowPane());
```

Note that the bottom region of the border pane is not used in this sample. If you want to add something to the bottom region, use the following statement and replace *node* with the control of your choice:

```
border.setBottom(node);
```

HBox

The `HBox` layout pane provides an easy way for arranging a series of nodes in a single row. Figure 1–2 shows an example of an `HBox` pane.

Figure 1–2 Sample HBox Pane

The padding property can be set to manage the distance between the nodes and the edges of the `HBox` pane. Spacing can be set to manage the distance between the nodes. The style can be set to change the background color.

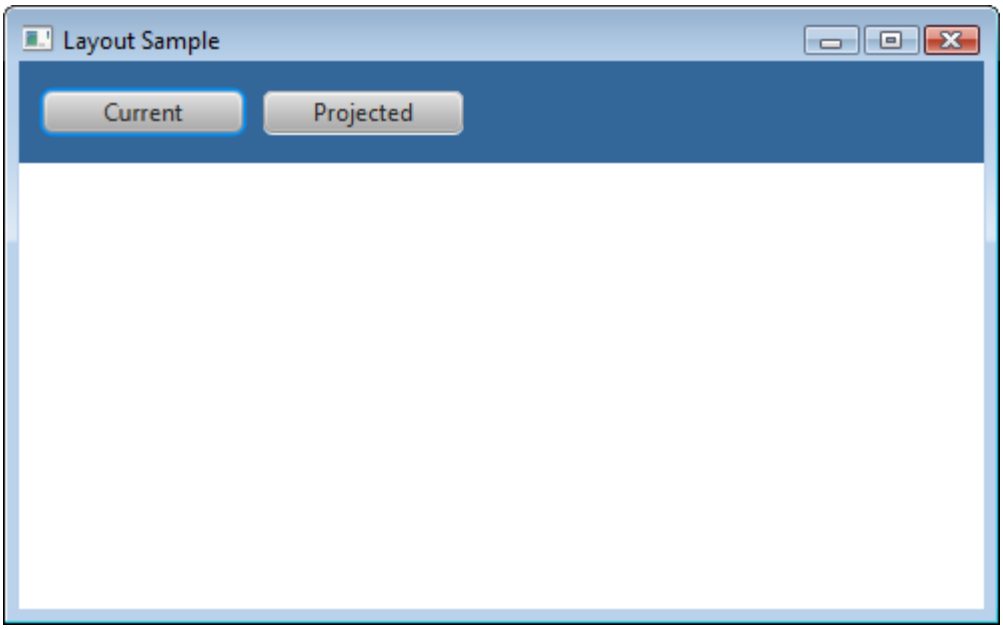
Example 1–2 creates an `HBox` pane for a tool bar that contains two buttons.

Example 1–2 Create an HBox Pane

```
public HBox addHBox() {
    HBox hbox = new HBox();
    hbox.setPadding(new Insets(15, 12, 15, 12));
    hbox.setSpacing(10);
    hbox.setStyle("-fx-background-color: #336699;");

    Button buttonCurrent = new Button("Current");
    buttonCurrent.setPrefSize(100, 20);

    Button buttonProjected = new Button("Projected");
    buttonProjected.setPrefSize(100, 20);
    hbox.getChildren().addAll(buttonCurrent, buttonProjected);

    return hbox;
}
```

The `setTop()` method in Example 1–1 adds the `HBox` pane to the top region of the border pane. The result is shown in Figure 1–3.

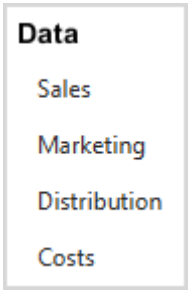
Figure 1–3 HBox Pane in a Border Pane



VBox

The VBox layout pane is similar to the HBox layout pane, except that the nodes are arranged in a single column. Figure 1–4 shows an example of a VBox pane.

Figure 1–4 Sample VBox Pane



The padding property can be set to manage the distance between the nodes and the edges of the VBox pane. Spacing can be set to manage the distance between the nodes. Margins can be set to add additional space around individual controls.

Example 1–3 creates a VBox pane for a list of options.

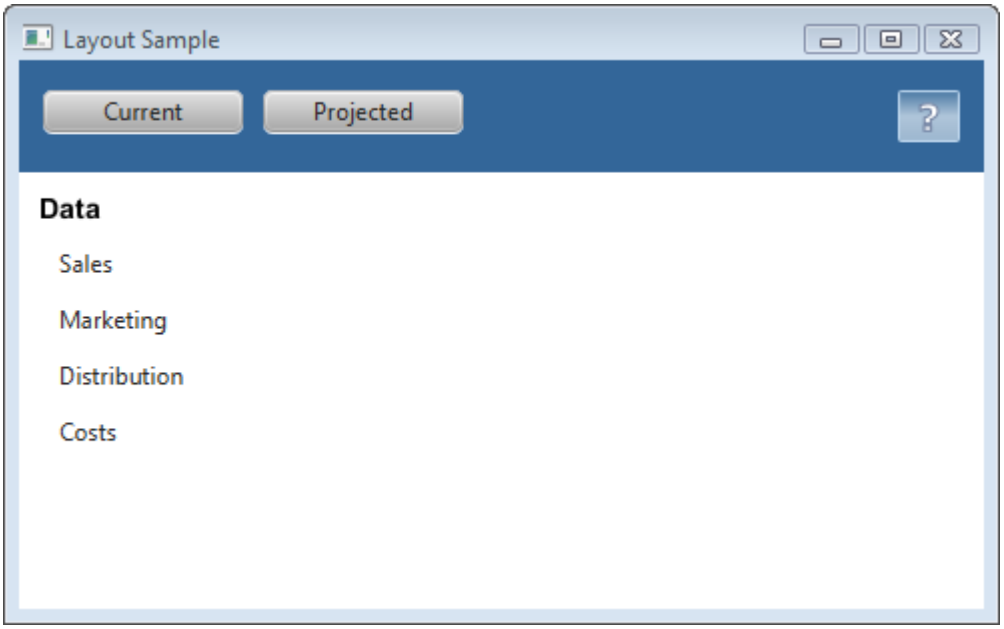
Example 1–3 Create a VBox Pane

```
public VBox addVBox(); {
    VBox vbox = new VBox();
    vbox.setPadding(new Insets(10));
    vbox.setSpacing(8);

    Text title = new Text("Data");
    title.setFont(Font.font("Arial", FontWeight.BOLD, 14));
    vbox.getChildren().add(title);
```

```
    Hyperlink options[] = new Hyperlink[] {
        new Hyperlink("Sales"),
        new Hyperlink("Marketing"),
        new Hyperlink("Distribution"),
        new Hyperlink("Costs")};

    for (int i=0; i<4; i++) {
        VBox.setMargin(options[i], new Insets(0, 0, 0, 8));
        vbox.getChildren().add(options[i]);
    }

    return vbox;
}
```

The `setLeft()` method in Example 1–1 adds the `VBox` pane to the left region of the border pane. The result is shown in Figure 1–5.

Figure 1–5 VBox Pane in a Border Pane



StackPane

The `StackPane` layout pane places all of the nodes within a single stack with each new node added on top of the previous node. This layout model provides an easy way to overlay text on a shape or image or to overlap common shapes to create a complex shape. Figure 1–6 shows a help icon that is created by stacking a question mark on top of a rectangle with a gradient background.

Figure 1–6 Sample Stack Pane

The alignment property can be set to manage how children are positioned in the stack pane. This property affects all children, so margins can be set to adjust the position of individual children in the stack.

Example 1–4 creates a stack pane for a help icon.

Example 1–4 Create a Stack Pane

```
public void addStackPane(HBox hb) {
    StackPane stack = new StackPane();
    Rectangle helpIcon = new Rectangle(30.0, 25.0);
    helpIcon.setFill(new LinearGradient(0,0,0,1, true, CycleMethod.NO_CYCLE,
        new Stop[]{
        new Stop(0,Color.web("#4977A3")),
        new Stop(0.5, Color.web("#B0C6DA")),
        new Stop(1,Color.web("#9CB6CF")),}));
    helpIcon.setStroke(Color.web("#D0E6FA"));
    helpIcon.setArcHeight(3.5);
    helpIcon.setArcWidth(3.5);

    Text helpText = new Text("?");
    helpText.setFont(Font.font("Verdana", FontWeight.BOLD, 18));
    helpText.setFill(Color.WHITE);
    helpText.setStroke(Color.web("#7080A0"));

    stack.getChildren().addAll(helpIcon, helpText);
    stack.setAlignment(Pos.CENTER_RIGHT);     // Right-justify nodes in stack
    StackPane.setMargin(helpText, new Insets(0, 10, 0, 0)); // Center "?"

    hb.getChildren().add(stack);            // Add to HBox from Example 1-2
    HBox.setHgrow(stack, Priority.ALWAYS);    // Give stack any extra space
}
```

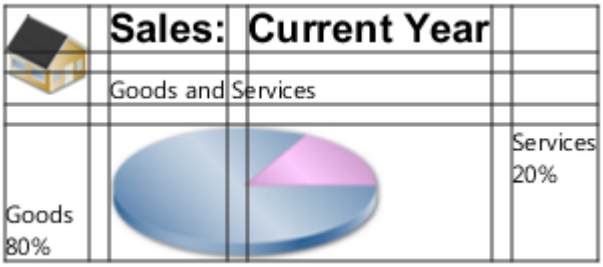
The last lines of code in Example 1–4 add the stack pane to the `HBox` pane created in Example 1–2 and position it to always be at the right-most edge of the pane. The result is shown in Figure 1–7.

Figure 1–7 Stack Pane in an HBox Pane

GridPane

The `GridPane` layout pane enables you to create a flexible grid of rows and columns in which to lay out nodes. Nodes can be placed in any cell in the grid and can span cells as needed. A grid pane is useful for creating forms or any layout that is organized in rows and columns. Figure 1–8 shows a grid pane that contains an icon, title, subtitle, text and a pie chart. In this figure, the `gridLinesVisible` property is set to display grid lines, which show the rows and columns and the gaps between the rows and columns. This property is useful for visually debugging your `GridPane` layouts.

Figure 1–8 Sample Grid Pane



Gap properties can be set to manage the spacing between the rows and columns. The padding property can be set to manage the distance between the nodes and the edges of the grid pane. The vertical and horizontal alignment properties can be set to manage the alignment of individual controls in a cell.

Example 1–5 creates the grid pane shown in Figure 1–8.

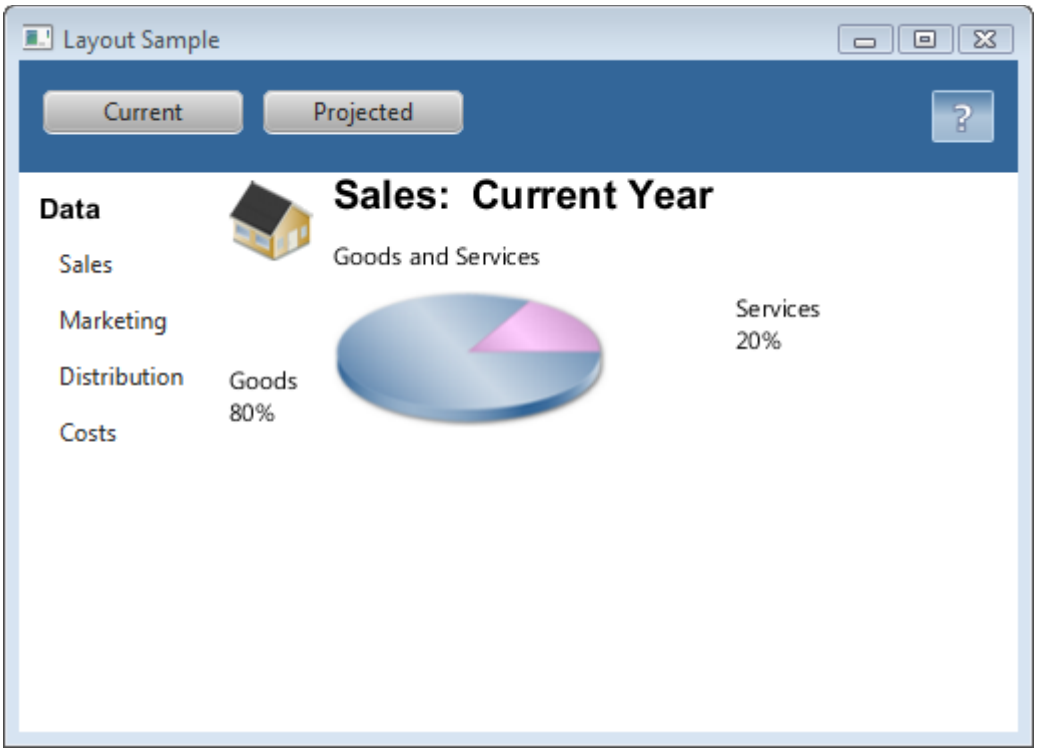
Example 1–5 Create a Grid Pane

```
public GridPane addGridPane() {
    GridPane grid = new GridPane();
    grid.setHgap(10);
    grid.setVgap(10);
    grid.setPadding(new Insets(0, 10, 0, 10));

    // Category in column 2, row 1
    Text category = new Text("Sales:");
    category.setFont(Font.font("Arial", FontWeight.BOLD, 20));
    grid.add(category, 1, 0);

    // Title in column 3, row 1
    Text chartTitle = new Text("Current Year");
    chartTitle.setFont(Font.font("Arial", FontWeight.BOLD, 20));
    grid.add(chartTitle, 2, 0);

    // Subtitle in columns 2-3, row 2
    Text chartSubtitle = new Text("Goods and Services");
    grid.add(chartSubtitle, 1, 1, 2, 1);

    // House icon in column 1, rows 1-2
    ImageView imageHouse = new ImageView(
      new Image(LayoutSample.class.getResourceAsStream("graphics/house.png")));
    grid.add(imageHouse, 0, 0, 1, 2);

    // Left label in column 1 (bottom), row 3
    Text goodsPercent = new Text("Goods\n80%");
    GridPane.setValignment(goodsPercent, VPos.BOTTOM);
    grid.add(goodsPercent, 0, 2);

    // Chart in columns 2-3, row 3
    ImageView imageChart = new ImageView(
     new Image(LayoutSample.class.getResourceAsStream("graphics/piechart.png")));
    grid.add(imageChart, 1, 2, 2, 1);

    // Right label in column 4 (top), row 3
    Text servicesPercent = new Text("Services\n20%");
    GridPane.setValignment(servicesPercent, VPos.TOP);
    grid.add(servicesPercent, 3, 2);
```

```
    return grid;
}
```

The `setCenter()` method in Example 1–1 adds the grid pane to the center region of the border pane. The result is shown in Figure 1–9.

Figure 1–9 Grid Pane in a Border Pane

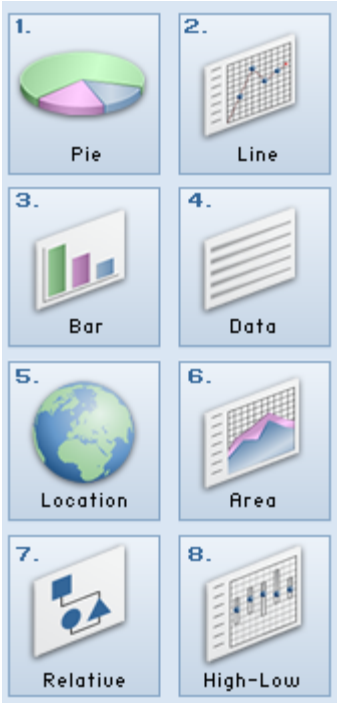


As the window is resized, the nodes within the grid pane are resized according to their layout constraints.

FlowPane

The nodes within a `FlowPane` layout pane are laid out consecutively and wrap at the boundary set for the pane. Nodes can flow vertically (in columns) or horizontally (in rows). A vertical flow pane wraps at the height boundary for the pane. A horizontal flow pane wraps at the width boundary for the pane. Figure 1–10 shows a sample horizontal flow pane using numbered icons. By contrast, in a vertical flow pane, column one would contain pages one through four and column two would contain pages five through eight.

Figure 1–10 Sample Horizontal Flow Pane



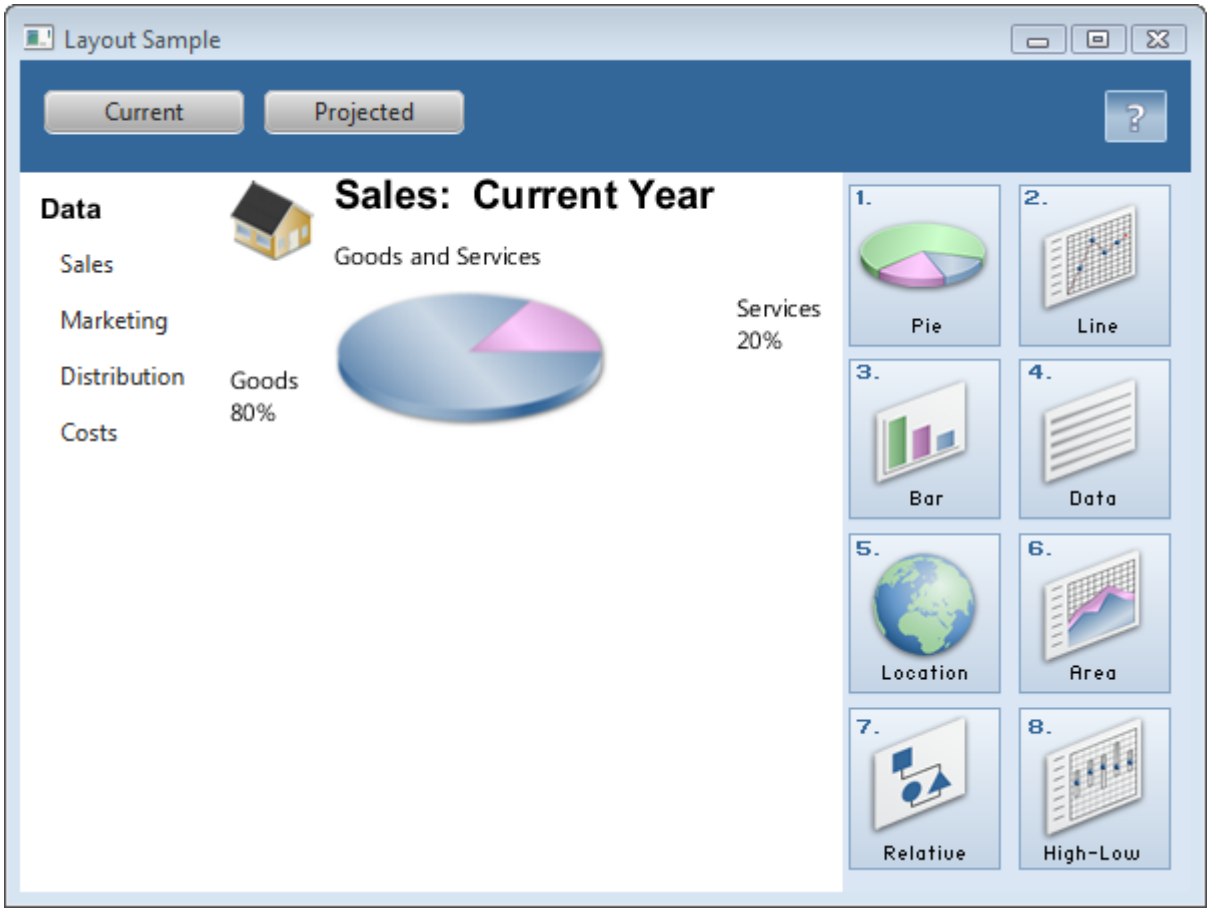
Gap properties can be set to manage the spacing between the rows and columns. The padding property can be set to manage the distance between the nodes and the edges of the pane. Example 1–6 creates a horizontal flow pane for a series of page icons.

Example 1–6 Create a Flow Pane

```
public FlowPane addFlowPane() {
    FlowPane flow = new FlowPane();
    flow.setPadding(new Insets(5, 0, 5, 0));
    flow.setVgap(4);
    flow.setHgap(4);
    flow.setPrefWrapLength(170); // preferred width allows for two columns
    flow.setStyle("-fx-background-color: DAE6F3;");

    ImageView pages[] = new ImageView[8];
    for (int i=0; i<8; i++) {
        pages[i] = new ImageView(
            new Image(LayoutSample.class.getResourceAsStream(
            "graphics/chart_"+(i+1)+".png")));
        flow.getChildren().add(pages[i]);
    }

    return flow;
}
```

The `setRight()` method in Example 1–1 adds the flow pane to the right region of the border pane. The result is shown in Figure 1–11.

Figure 1–11 Flow Pane in a Border Pane



TilePane

A tile pane is similar to a flow pane. The `TilePane` layout pane places all of the nodes in a grid in which each cell, or tile, is the same size. Nodes can be laid out horizontally (in rows) or vertically (in columns). Horizontal tiling wraps the tiles at the tile pane's width boundary and vertical tiling wraps them at the height boundary. Use the `prefColumns` and `prefRows` properties to establish the preferred size of the tile pane.

Gap properties can be set to manage the spacing between the rows and columns. The padding property can be set to manage the distance between the nodes and the edges of the pane.

Example 1–7 creates a horizontal tile pane that produces the same layout shown in Figure 1–10.

Example 1–7 Create a Tile Pane

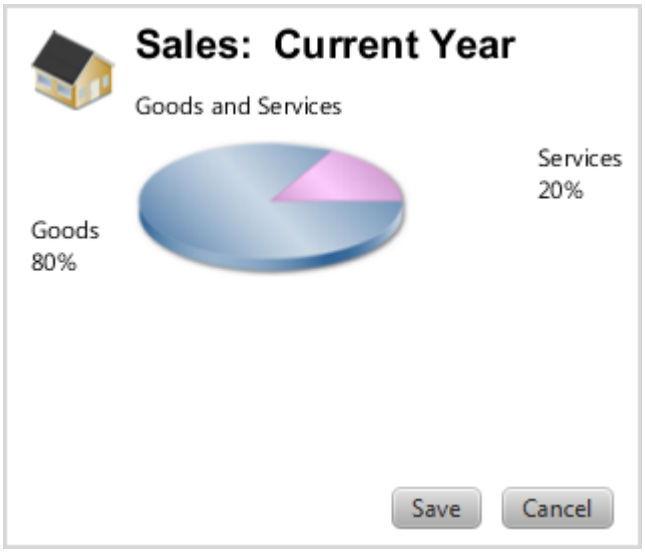
```
TilePane tile = new TilePane();
tile.setPadding(new Insets(5, 0, 5, 0));
tile.setVgap(4);
tile.setHgap(4);
tile.setPrefColumns(2);
tile.setStyle("-fx-background-color: DAE6F3;");

ImageView pages[] = new ImageView[8];
```

```
for (int i=0; i<8; i++) {
     pages[i] = new ImageView(
        new Image(LayoutSample.class.getResourceAsStream(
        "graphics/chart_"+(i+1)+".png")));
     tile.getChildren().add(pages[i]);
}
```

AnchorPane

The `AnchorPane` layout pane enables you to anchor nodes to the top, bottom, left side, right side, or center of the pane. As the window is resized, the nodes maintain their position relative to their anchor point. Nodes can be anchored to more than one position and more than one node can be anchored to the same position. Figure 1–12 shows an anchor pane with the grid pane from GridPane anchored to the top and an `HBox` pane with two buttons anchored to the bottom and the right side.

Figure 1–12 Sample Anchor Pane



Example 1–8 creates an anchor pane with a node anchored to the top of the pane and a node anchored to the bottom right of the pane. The grid that was created in Example 1–5 is used for the top node.

Example 1–8 Create an Anchor Pane

```
public AnchorPane addAnchorPane(GridPane grid) {
    AnchorPane anchorpane = new AnchorPane();
    Button buttonSave = new Button("Save");
    Button buttonCancel = new Button("Cancel");

    HBox hb = new HBox();
    hb.setPadding(new Insets(0, 10, 10, 10));
    hb.setSpacing(10);
    hb.getChildren().addAll(buttonSave, buttonCancel);

    anchorpane.getChildren().addAll(grid,hb);   // Add grid from Example 1-5
    AnchorPane.setBottomAnchor(hb, 8.0);
    AnchorPane.setRightAnchor(hb, 5.0);
    AnchorPane.setTopAnchor(grid, 10.0);
```

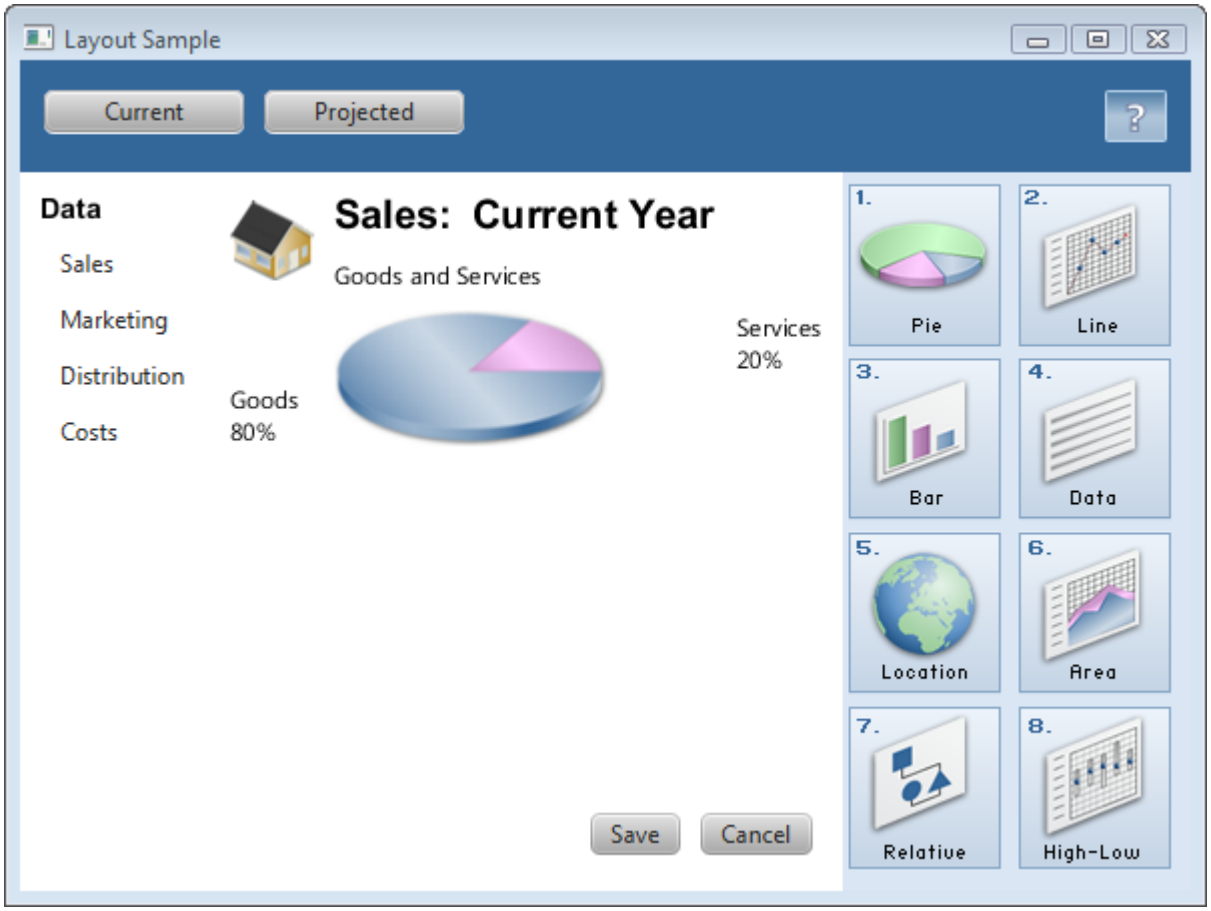
```
    return anchorpane;
}
```

The following statement replaces the center region of the border pane with the anchor pane:

```
border.setCenter(addAnchorPane(addGridPane()));
```

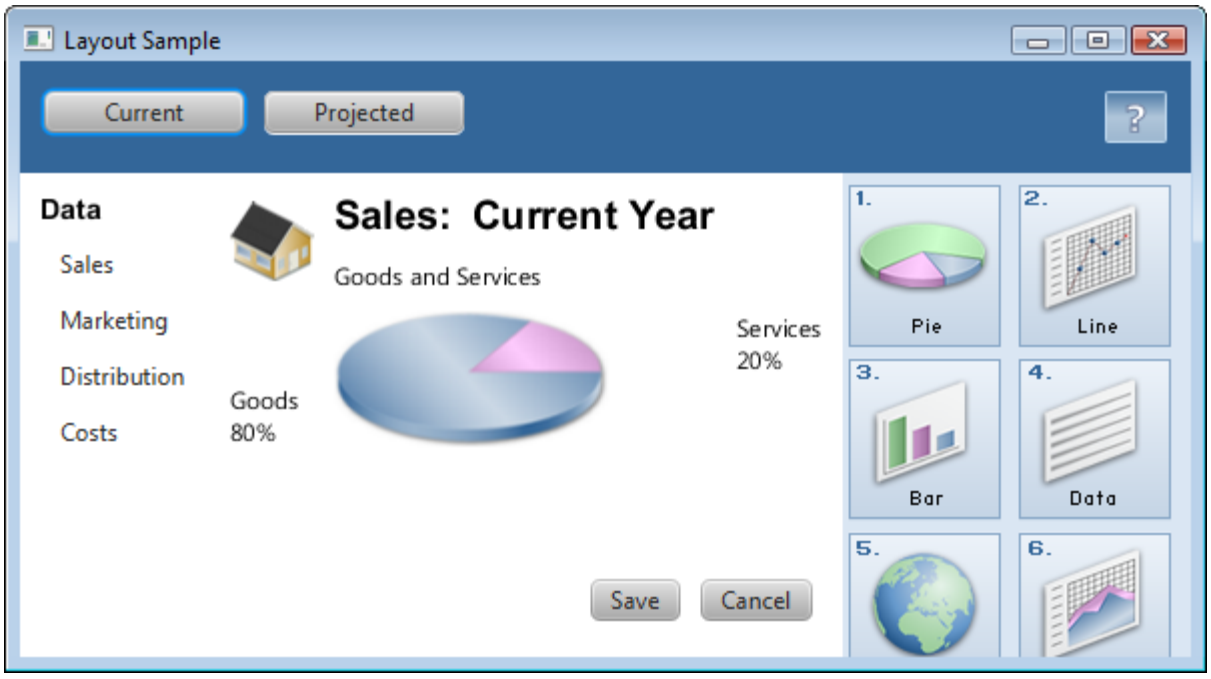
The result is shown in Figure 1–13.

Figure 1–13 Anchor Pane in a Border Pane



As the window is resized, the nodes maintain their position in the pane according to their anchor points. Figure 1–14 shows how the buttons, which are anchored to the bottom of the pane, move closer to the sales information as the window is made smaller.

Figure 1–14 Resized Anchor Pane



Additional Resources

To learn more about the layout panes in JavaFX, see the information for the `javafx.scene.layout` package in the API Documentation.

2

Tips for Sizing and Aligning Nodes

This topic describes techniques for controlling the size and alignment of nodes when placed in a JavaFX layout pane.

A main advantage of using the built-in JavaFX layout panes is that the size and alignment of nodes is handled by the pane. As the pane is resized, the nodes are resized according to their preferred size range preferences. Note that not all node classes are resizable. UI controls and layout panes are resizable, but shapes, `Text` objects, and `Group` objects are not resizable and are treated as rigid objects in a layout.

If you want more control over the size of controls in your UI, you can set their preferred size range directly. The layout pane then uses your settings to determine the size of the control. To manage the position of the controls, you can use the alignment properties for the layout panes.

This topic provides simple examples for sizing and aligning nodes in a pane. The LayoutSizingAligning.java file contains the source code for the samples described in this topic. The LayoutSizingAligning.zip file contains the NetBeans IDE project for the sample.

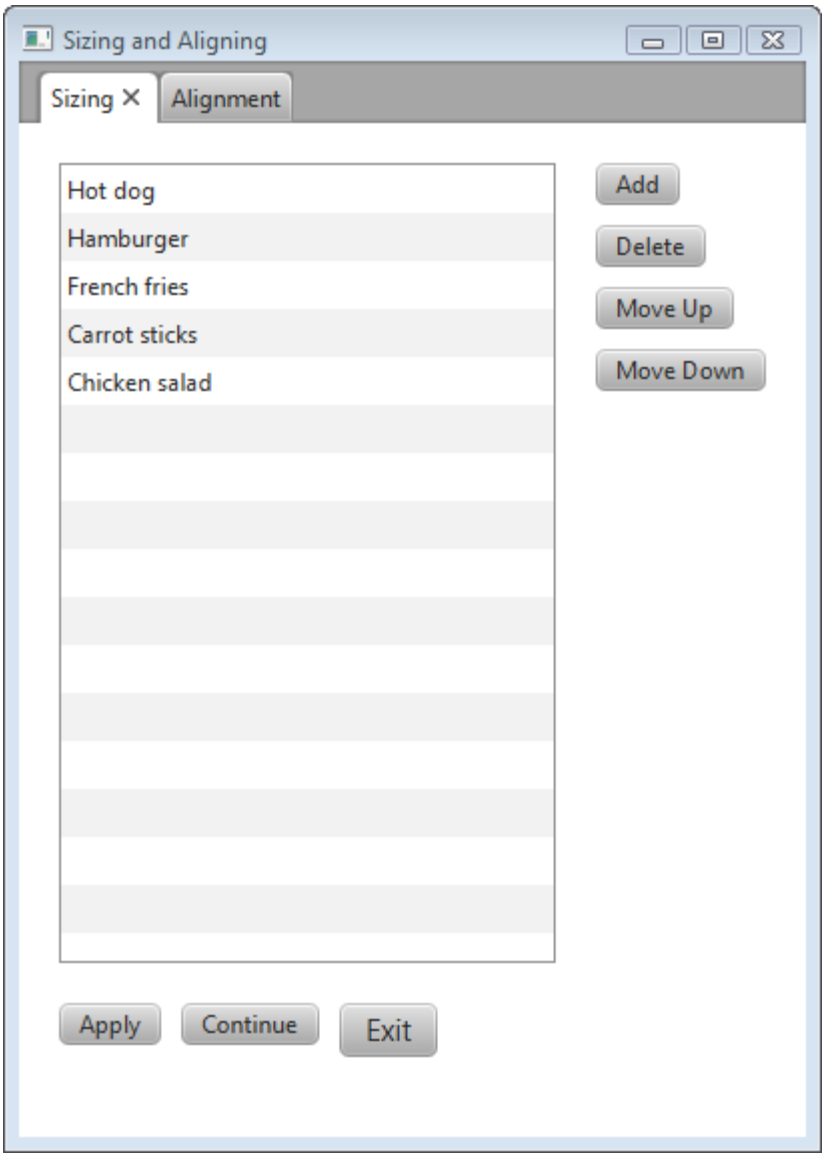
Sizing Nodes

Layouts query the preferred size of their nodes by invoking the `prefWidth(height)` and `prefHeight(width)` methods. By default, UI controls compute default values for their preferred size that is based on the content of the control. For example, the computed size of a `Button` object is determined by the length of the text and the size of the font used for the label, plus the size of any image. Typically, the computed size is just big enough for the control and the label to be fully visible.

UI controls also provide default minimum and maximum sizes that are based on the typical usage of the control. For example, the maximum size of a `Button` object defaults to its preferred size because you don't usually want buttons to grow arbitrarily large. However, the maximum size of a `ScrollPane` object is unbounded because typically you do want them to grow to fill their spaces.

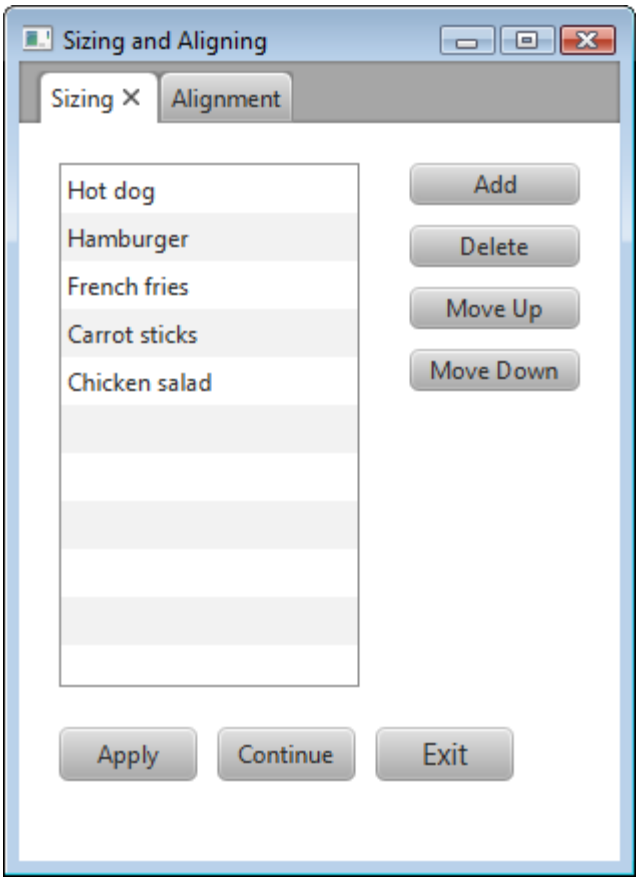
You can use the default size constraints of nodes, or you can set them to provide the look that you want. For example, Figure 2–1 shows the default size of several buttons and a list view in a border pane.

Figure 2–1 Computed Sizes



Suppose that the look that you want is the screen shown in Figure 2–2, which shows the UI controls sized according to desired constraints.

Figure 2–2 Desired Sizes



Applications often need to directly set the minimum, preferred, and maximum size constraints on controls. The following sections provide tips for overriding the computed sizes to get the look that you want.

Making Buttons the Same Size

You can go through the trouble of determining the height and width of each button and then setting the preferred size of each button to the greatest height and width of the buttons in the set. An easier option is to let the layout panes do the work. The layout pane that you want to use is determined by the effect that you want to achieve.

Using a VBox

The scene in Figure 2–1 uses a `VBox` layout pane for the buttons on the right and uses the computed sizes for the buttons. The buttons already have the same height, so only the width needs to be changed.

The scene in Figure 2–2 uses a `VBox` pane to take advantage of the default behavior that makes the width of the `VBox` pane the same as the preferred width of its widest element. To enable all of the buttons to be resized to the width of the `VBox` pane, the maximum width of each button is set to the `Double.MAX_VALUE` constant, which enables a control to grow without limit. An alternative to using the maximum value constant is to set the maximum width to a specific value, such as 80.0.

Example 2–1 shows how to make a column of buttons the same width using a `VBox` pane.

Example 2–1 Set a Column of Buttons to the Same Width

```
BorderPane border = new BorderPane();
border.setPadding(new Insets(20, 0, 20, 20));

Button btnAdd = new Button("Add");
Button btnDelete = new Button("Delete");
Button btnMoveUp = new Button("Move Up");
Button btnMoveDown = new Button("Move Down");

btnAdd.setMaxWidth(Double.MAX_VALUE);
btnDelete.setMaxWidth(Double.MAX_VALUE);
btnMoveUp.setMaxWidth(Double.MAX_VALUE);
btnMoveDown.setMaxWidth(Double.MAX_VALUE);

VBox vbButtons = new VBox();
vbButtons.setSpacing(10);
vbButtons.setPadding(new Insets(0, 20, 10, 20));
vbButtons.getChildren().addAll(btnAdd, btnDelete, btnMoveUp, btnMoveDown);
```

In the Layout Sizing and Aligning sample, the elements of the UI are laid out using a border pane. The column of buttons is put in the right region of the border pane to limit the size of the buttons to the preferred width of the widest button. The center region of a border pane expands to fill any space available so if you put the `VBox` pane in the center region, the `VBox` pane and the buttons also expand.

Using a TilePane

The scene in Figure 2–1 uses an `HBox` layout pane for the buttons on the bottom and uses the computed sizes for those buttons. The buttons have different widths and heights.

The scene in Figure 2–2 uses a horizontal `TilePane` layout pane to take advantage of the default behavior that makes each cell (tile) the same size. The size of each tile is the size needed to hold the preferred size of the largest node in the tile pane.

To enable the buttons to be resized to the size of the tile, set the maximum width and height to the `Double.MAX_VALUE` constant. Example 2–2 shows how make a row of buttons the same width and height using a tile pane.

Example 2–2 Set a Row of Buttons to the Same Size

```
Button btnApply = new Button("Apply");
Button btnContinue = new Button("Continue");
Button btnExit = new Button("Exit");
btnExit.setStyle("-fx-font-size: 15pt;");

btnApply.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
btnContinue.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
btnExit.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);

TilePane tileButtons = new TilePane(Orientation.HORIZONTAL);
tileButtons.setPadding(new Insets(20, 10, 20, 0));
tileButtons.setHgap(10.0);
tileButtons.setVGap(8.0);
tileButtons.getChildren().addAll(btnApply, btnContinue, btnExit);
```

Tiles are not resized as the window size changes so the buttons don't change size when set in a tile pane. Note that if the width of the window is reduced, the buttons in the tile pane change position, but do not get smaller.

Keeping Nodes at Their Preferred Size

As a stage is resized, layout panes within the stage might have more or less space to allocate to the controls that they contain. Each layout pane has its own rules for allocating space according to the minimum, preferred, and maximum size ranges of the controls.

In general, controls that have a default maximum size of `Double.MAX_VALUE` expand to fill their space while controls with constrained maximum sizes do not expand. For example, a `ListView` object has an unbounded maximum. If you want to limit the height to its preferred size, you can set the maximum size to the `Control.USE_PREF_SIZE` constant, as shown in Example 2–3.

Example 2–3 Set Maximum Height to Preferred Height

```
ListView<String> lvList = new ListView<String>();
ObservableList<String> items = FXCollections.observableArrayList (
        "Hot dog", "Hamburger", "French fries",
        "Carrot sticks", "Chicken salad");
lvList.setItems(items);
lvList.setMaxHeight(Control.USE_PREF_SIZE);
```

By default, buttons grow only to their preferred size. However, buttons shrink to where the label is shown as three dots (...) if the minimum width is not overridden. To prevent a button from becoming smaller than its preferred width, set its minimum width to its preferred width as shown in Example 2–4.

Example 2–4 Set Minimum Width to Preferred Width

```
Button btnMoveDown = new Button("Move Down");
btnMoveDown.setMinWidth(Control.USE_PREF_SIZE);
```

The preferred size of a control is initially based on the computed size. You can override the default preferred size by setting the preferred size constraint to the size of your choice. The following statement overrides the preferred width of a list view:

```
lvList.setPrefWidth(150.0);
```

Preventing Resizing

If you do not want the size of a node to change, set the minimum, maximum, and preferred sizes to the same size. To prevent only the width or height from changing, set the width or height constraints to the same value. In Example 2–5, a list is created with all size constraints set to the same width and height values so that the size of the list doesn't change as the size of the window changes. A button is created with all width constraints set to the same value.

Example 2–5 Set Size Constraints to Prevent Resizing

```
ListView<String> lvList = new ListView<String>();
lvList.setMinSize(150.0, Control.USE_PREF_SIZE);
lvList.setMaxSize(150.0, Control.USE_PREF_SIZE);

Button btnDelete = new Button("Delete");
btnDelete.setMinWidth(80.0);
btnDelete.setPrefWidth(80.0);
btnDelete.setMaxWidth(80.0);
```

Aligning Content

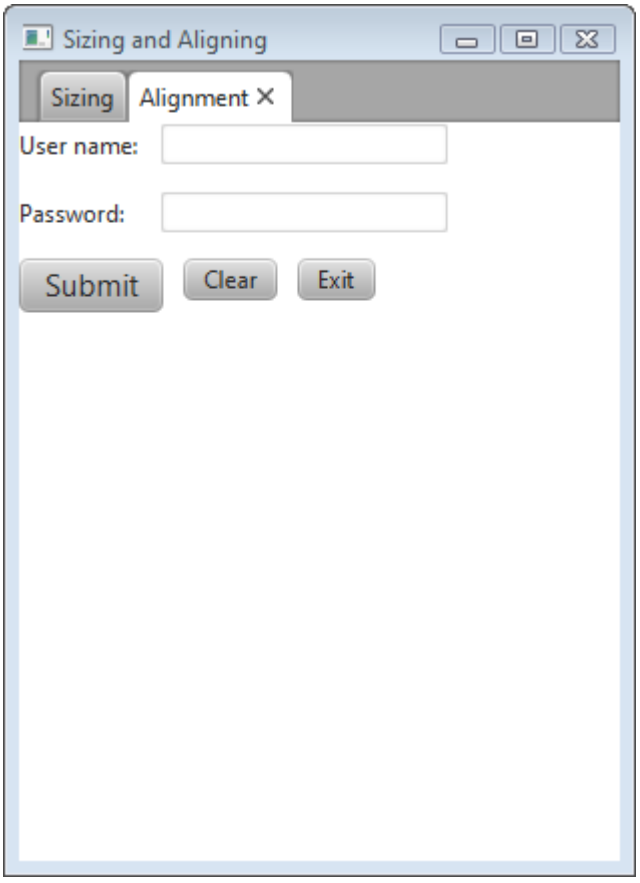
Each layout pane has a default way of aligning the nodes within the pane. For example, in `HBox` and `VBox` layout panes, nodes are top-justified and left-justified. In `TilePane` and `FlowPane` layout panes, nodes are centered. The panes themselves by default are typically top-justified and left-justified.

You can manage the alignment of nodes and panes by using the `setAlignment()` method for the panes. Alignment constants are available in the following `enum` types in the `javafx.geometry` package:

- `HPos` - Values for specifying horizontal alignment.
- `Pos` - Values for specifying vertical and horizontal alignment. The value to the left of the underscore specifies the vertical alignment, the value to the right of the underscore specifies the horizontal alignment. For example, `Pos.BOTTOM_LEFT` aligns a node at the bottom of the space vertically and at the left edge of the space horizontally.
- `VPos` - Values for specifying vertical alignment.

Figure 2–3 is created by the code shown in Example 2–6. Without any alignment constraints specified, the layout pane is placed in the top left corner.

Figure 2–3 Default Positions



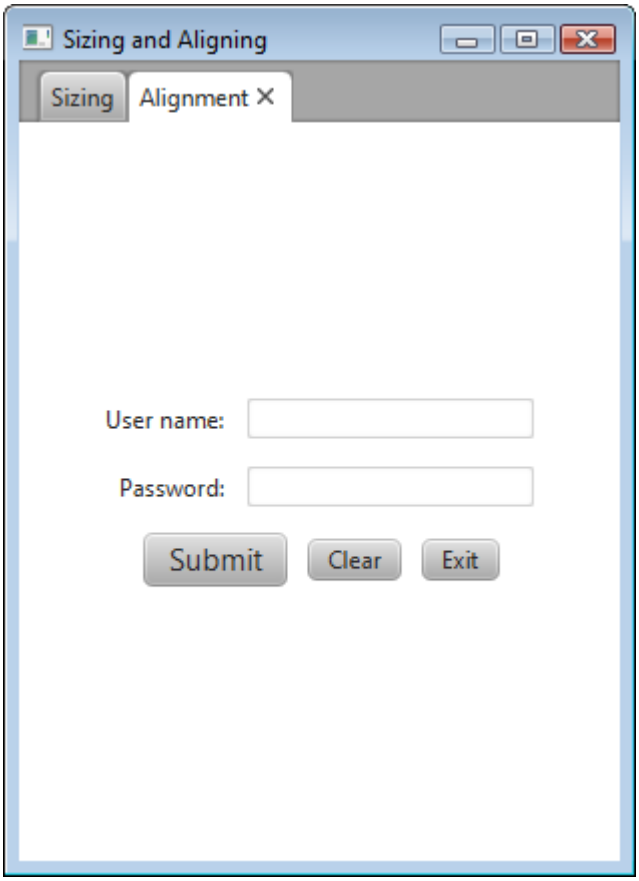
Example 2–6 Create a UI with Default Alignment

```
GridPane grid = new GridPane();
grid.setHgap(10);
```

```
        grid.setVgap(12);

        HBox hbButtons = new HBox();
        hbButtons.setSpacing(10.0);

        Button btnSubmit = new Button("Submit");
        Button btnClear = new Button("Clear");
        Button btnExit = new Button("Exit");
        btnSubmit.setStyle("-fx-font-size: 15pt;");

        Label lblName = new Label("User name:");
        TextField tfName = new TextField();
        Label lblPwd = new Label("Password:");
        PasswordField pfPwd = new PasswordField();

        hbButtons.getChildren().addAll(btnSubmit, btnClear, btnExit);
        grid.add(lblName, 0, 0);
        grid.add(tfName, 1, 0);
        grid.add(lblPwd, 0, 1);
        grid.add(pfPwd, 1, 1);
        grid.add(hbButtons, 0, 2, 2, 1);
    }
```

Suppose the look that you want is the screen shown in Figure 2–4, which centers the layout pane in the screen and changes the default alignment of the controls.

Figure 2–4 Desired Positions



The following sections provide tips for overriding the default positions.

Centering the Grid

To center the grid from Example 2–6 in the scene, use the following statement:

```
grid.setAlignment(Pos.CENTER);
```

Aligning Controls in the Columns

In the desired layout, the labels are right-justified and the fields are left-justified. To achieve this in a grid, define each column using the `ColumnConstraints` class and set the horizontal alignment constraint. Example 2–7 defines the columns for the grid from Example 2–6.

Example 2–7 Define the Columns in the Grid

```
GridPane grid = new GridPane();
grid.setAlignment(Pos.CENTER);
grid.setHgap(10);
grid.setVgap(12);

ColumnConstraints column1 = new ColumnConstraints();
column1.setHalignment(HPos.RIGHT);
grid.getColumnConstraints().add(column1);

ColumnConstraints column2 = new ColumnConstraints();
column2.setHalignment(HPos.LEFT);
grid.getColumnConstraints().add(column2);
```

An alternative for right-justifying controls in a column is to use a `VBox` layout pane. Use the `setAlignment()` method as shown in the following statements:

```
VBox vbox = new VBox;
vbox.setAlignment(Pos.CENTER_RIGHT);
```

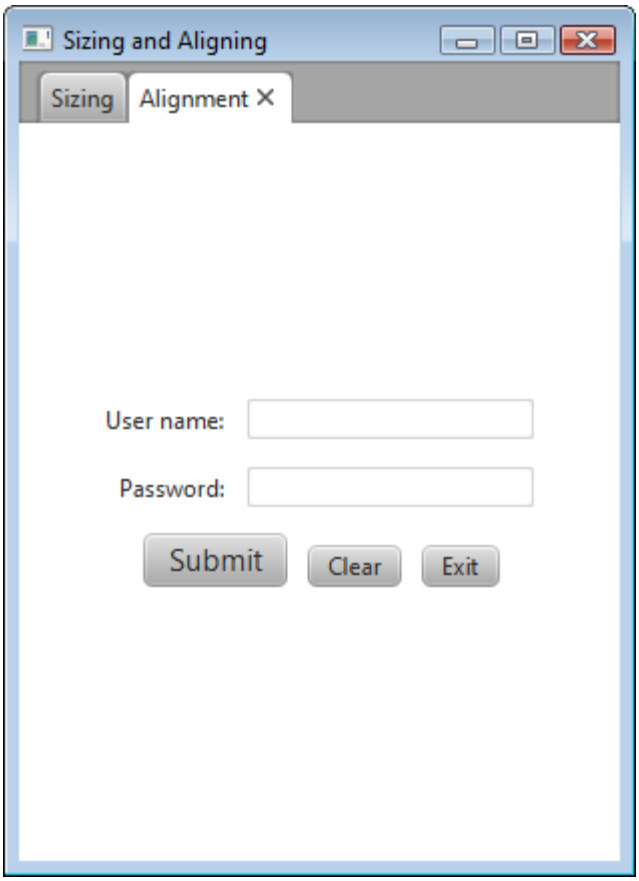
Centering the Buttons

The buttons are laid out in an `HBox` layout pane that spans both columns in the grid. The following statement centers the buttons in the grid from Example 2–6:

```
hbButtons.setAlignment(Pos.CENTER);
```

The `setAlignment()` method for the `HBox` pane centers the `HBox` pane within its layout area and also centers the nodes within the `HBox` pane. You might prefer to center the `HBox` pane within the row but bottom-justify the buttons within the `HBox` pane as shown in Figure 2–5.

Figure 2–5 Override Positions and Bottom-Justify the Buttons



For this arrangement, place the `HBox` pane in an inner grid with a single cell and place that grid in the third row of the outer grid. Set the alignment constraint for the inner grid to center the grid and set the alignment constraint for the `HBox` pane to bottom-justify the contents as shown in Example 2–8.

Example 2–8 Center and Bottom-Justify the Buttons

```
hbButtons.setAlignment(Pos.BOTTOM_CENTER);
hbButtons.getChildren().addAll(btnSubmit, btnClear, btnExit);

GridPane innergrid = new GridPane();
innergrid.setAlignment(Pos.CENTER);
innergrid.add(hbButtons, 0, 0);
grid.add(innergrid, 0, 2, 2, 1);
```

Additional Resources

To learn more about the layout panes in JavaFX, see the information for the `javafx.scene.layout` package in the API Documentation.

3

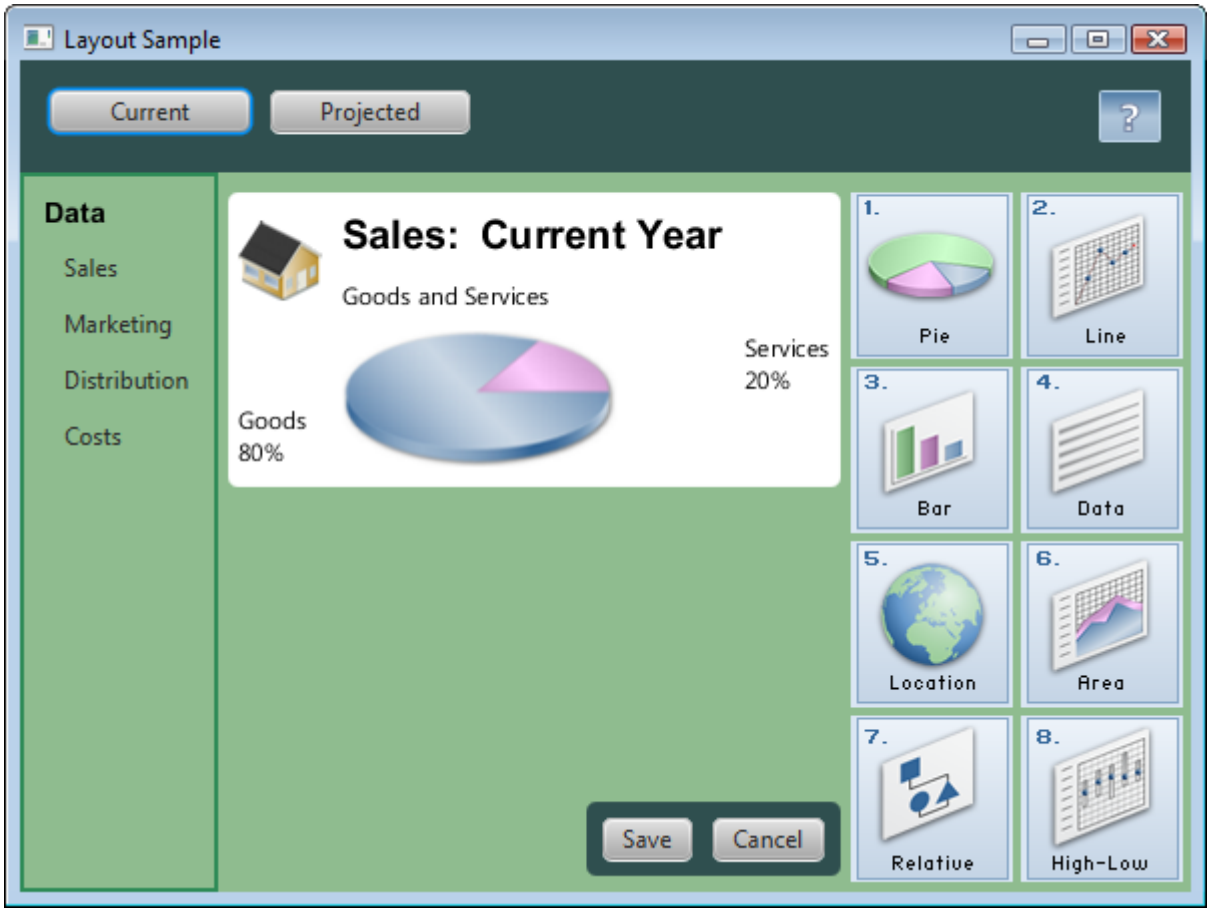
Styling Layout Panes with CSS

This topic describes how use CSS to style the layout panes that are available with the JavaFX SDK.

Layout panes use properties such as padding, spacing, and alignment to manage elements of how the panes appear. Cascading style sheets (CSS) enable you to define a set of properties and assign them to multiple layout panes to provide a standard look to your JavaFX application. You can also use CSS to customize individual layout panes.

This topic uses the sample from Using Built-in Layout Panes to provide examples of styling the different layout panes with CSS. Figure 3–1 shows the new look that is created.

Figure 3–1 Layout Sample with Custom Styling



The LayoutSampleCSS.java file contains the source code for building this user interface. The LayoutSampleCSS.zip file contains the NetBeans IDE project for the sample application.

Creating Style Definitions

Unlike controls such as the button and check box, which have the respective style classes `.button` and `.check-box`, no style classes are predefined for the layout panes. To style your layout panes, you must create a style sheet and define the style classes that you want. Then in the code for your application, assign the appropriate style class to layout panes as you create them.

For example, if you want all of your `HBox` panes to have the same background color, padding, and spacing properties, create a style named `.hbox` as shown in Example 3–1.

Example 3–1 Sample Style for an HBox Pane

```
.hbox {
    -fx-background-color: #2f4f4f;
    -fx-padding: 15;
    -fx-spacing: 10;
}
```

Use this style for each HBox pane that you create by assigning the style class to the pane. Example 3–2 shows the style assigned to two panes.

Example 3–2 Assigning the Style to HBox Panes

```
HBox hbox = new HBox();
hbox.getStyleClass().add("hbox");

HBox hb = new HBox();
hb.getStyleClass().add("hbox");
```

Style Properties for Layout Panes

You can use CSS to set the background, border, and padding properties for all types of layout panes. Some types of layout panes have additional properties that can be set. For example, you can set the spacing and alignment properties for HBox and VBox panes, and you can set the orientation, preferred number of rows, preferred number of columns, and other properties for tile panes. Images can be used for the background and border of a pane.

See the JavaFX CSS Reference Guide for a list of the properties that are available for each type of layout pane. Properties listed for the Region class can be used by all layout panes, which are descendents of the Region class.

Assigning a Style Sheet to the Scene

After you have your style sheet prepared, you must add the style sheet to the scene for your application. All nodes then have access to the styles defined in the style sheet. Example 3–3 shows how to add the style sheet for the Layout Sample. In this sample, the style sheet is in the same directory as the class file for the application.

Example 3–3 Adding a Style Sheet

```
Scene scene = new Scene(border);
scene.getStylesheets().add("layoutsamplecss/layoutstyles.css");
```

Styling the Layout Sample

The Layout Sample has examples of the built-in layout panes that are provided by the JavaFX layout package. Styling this sample provides examples of how CSS can be used with the different layout panes.

The style sheet layoutstyles.css contains the styles used for Figure 3–1.

Defining a Style for Shared Properties

All layout panes have a set of basic properties that can be managed with CSS. These include properties for the background, border, padding, and shape of the pane. If you have several panes that share common styling for these properties, you can define a single style class and assign that class to each of the panes.

In the customized Layout Sample, several of the layout panes use the same background color. The .pane style class shown in Example 3–4 is used to set this color.

Example 3–4 .pane Style Class

```
.pane {
    -fx-background-color: #8fbc8f;
}
```

If setting the background color is all that is needed for a pane, then only the `.pane` style needs to be assigned. If additional styling is desired, more than one style can be assigned to a pane. Example 3–5 shows just the `.pane` style added to an anchor pane and the `.pane` and `.grid` styles added to a grid pane.

Example 3–5 Assigning the .pane Style to a Layout Pane

```
AnchorPane anchorpane = new AnchorPane();
anchorpane.getStyleClass().add("pane");

GridPane grid = new GridPane();
grid.getStyleClass().addAll("pane","grid");
```

Styling the Border Pane

Border panes do not have any additional properties beyond the basic set mentioned in Defining a Style for Shared Properties. In the Layout Sample, the border pane is not styled. However, to style a border pane, define a style class and assign it to the pane in the same way that is done for other panes.

Styling the HBox Panes

In addition to the basic set of properties for all layout panes, `HBox` panes have properties for alignment, spacing, and fill height.

The Layout Sample shown in Figure 3–1 contains two `HBox` panes. The first `HBox` pane is at the top and contains the Current and Projected buttons. The second `HBox` pane is near the bottom and contains the Save and Cancel buttons.

For the customized Layout Sample, both `HBox` panes have the same background color and spacing. These properties are set in the style definition shown in Example 3–1.

The second `HBox`, which contains the Save and Cancel buttons, also has rounded corners and less padding as shown in Figure 3–2.

Figure 3–2 Styled HBox Pane

To use the defined styling for `HBox` panes, the `.hbox` style is assigned to the pane. To override the padding and set the additional property for rounding the corners, the style definition shown in Example 3–6 is used and an ID is set for the second `HBox` pane.

Example 3–6 Custom HBox Style

```
#hbox-custom {
    -fx-background-radius: 5.0;
    -fx-padding: 8;
}
```

Example 3–7 shows how the styles are assigned to the second HBox pane.

Example 3–7 Assigning the Custom HBox Style to a Pane

```
HBox hb = new HBox();
hb.getStyleClass().add("hbox");
hb.setId("hbox-custom");
```

Styling the VBox Pane

In addition to the basic set of properties for all layout panes, VBox panes have properties for alignment, spacing, and fill width.

The VBox pane in the left region of Figure 3–1 uses the background from the .pane style class. The border, padding, and spacing are set in the .vbox style class shown in Example 3–8.

Example 3–8 .vbox Style Class

```
.vbox {
    -fx-border-color: #2e8b57;
    -fx-border-width: 2px;
    -fx-padding: 10;
    -fx-spacing: 8;
}
```

Example 3–9 shows how the styles are assigned to the VBox pane.

Example 3–9 Assigning Styles to the VBox Pane

```
VBox vbox = new VBox();
vbox.getStyleClass().addAll("pane", "vbox");
```

Styling the Stack Pane

In addition to the basic set of properties for all layout panes, stack panes have a property for alignment. In the Layout Sample, the stack pane is not styled. However, to style a stack pane, define a style class and assign it to the pane in the same way that is done for other panes.

Styling the Grid Pane

In addition to the basic set of properties for all layout panes, grid panes have properties for spacing between the rows and columns, alignment of the grid, and visibility of the grid lines.

The grid in the center region of Figure 3–1 has rounded corners and a white background that is slightly smaller than the grid itself. The .grid style class shown in Example 3–10 provides this styling and sets the properties for padding and for spacing between the rows and columns.

Example 3–10 .grid Style Class

```
.grid {
    -fx-background-color: white;
    -fx-background-radius: 5.0;
    -fx-background-insets: 0.0 5.0 0.0 5.0;
    -fx-padding: 10;
```

```
    -fx-hgap: 10;
    -fx-vgap: 10;
}
```

Example 3–11 shows how the style is assigned to the grid.

Example 3–11 Assigning a Style to the Grid

```
GridPane grid = new GridPane();
grid.getStyleClass().add("grid");
```

Note that the grid does not have the background color that is used by other layout panes in the sample. However, the anchor pane that contains the grid uses the background color. To prevent the grid from having the background color of its parent, you must to set the background for the grid to the desired color.

Styling the Flow Pane or Tile Pane

In addition to the basic set of properties for all layout panes, flow panes have properties for alignment, orientation, and spacing between the rows and columns. Tile panes have properties for alignment, orientation, spacing between the rows and columns, preferred number of rows and columns, and preferred width and height.

In the Layout Sample, either a flow pane or a tile pane can be used for the right region of Figure 3–1. The properties set in the style class are common to both types of panes, so the same style class is used in the sample. The `.flow-tile` style class shown in Example 3–12 sets the properties for padding and for spacing between the rows and columns.

Example 3–12 .flow-tile Style Class

```
.flow-tile {
    -fx-padding: 10.0 3.0 5.0 0.0;
    -fx-hgap: 4;
    -fx-vgap: 4;
}
```

The flow pane and tile pane also use the `.pane` style class for the background color. Example 3–13 shows how the styles are assigned to the flow pane and tile pane.

Example 3–13 Assigning Styles to the Flow Pane and Tile Pane

```
FlowPane flow = new FlowPane();
flow.getStyleClass().addAll("pane", "flow-tile");

TilePane tile = new TilePane();
tile.getStyleClass().addAll("pane", "flow-tile");
```

Styling the Anchor Pane

Anchor panes do not have any additional properties beyond the basic set of properties for all layout panes.

The anchor pane in the center region of Figure 3–1 contains a grid and an `HBox` pane, each of which has its own styling. The only styling applied to the anchor pane is the background color set by the `.pane` style class shown in Example 3–4.

Example 3–14 shows how the style is assigned to the anchor pane.

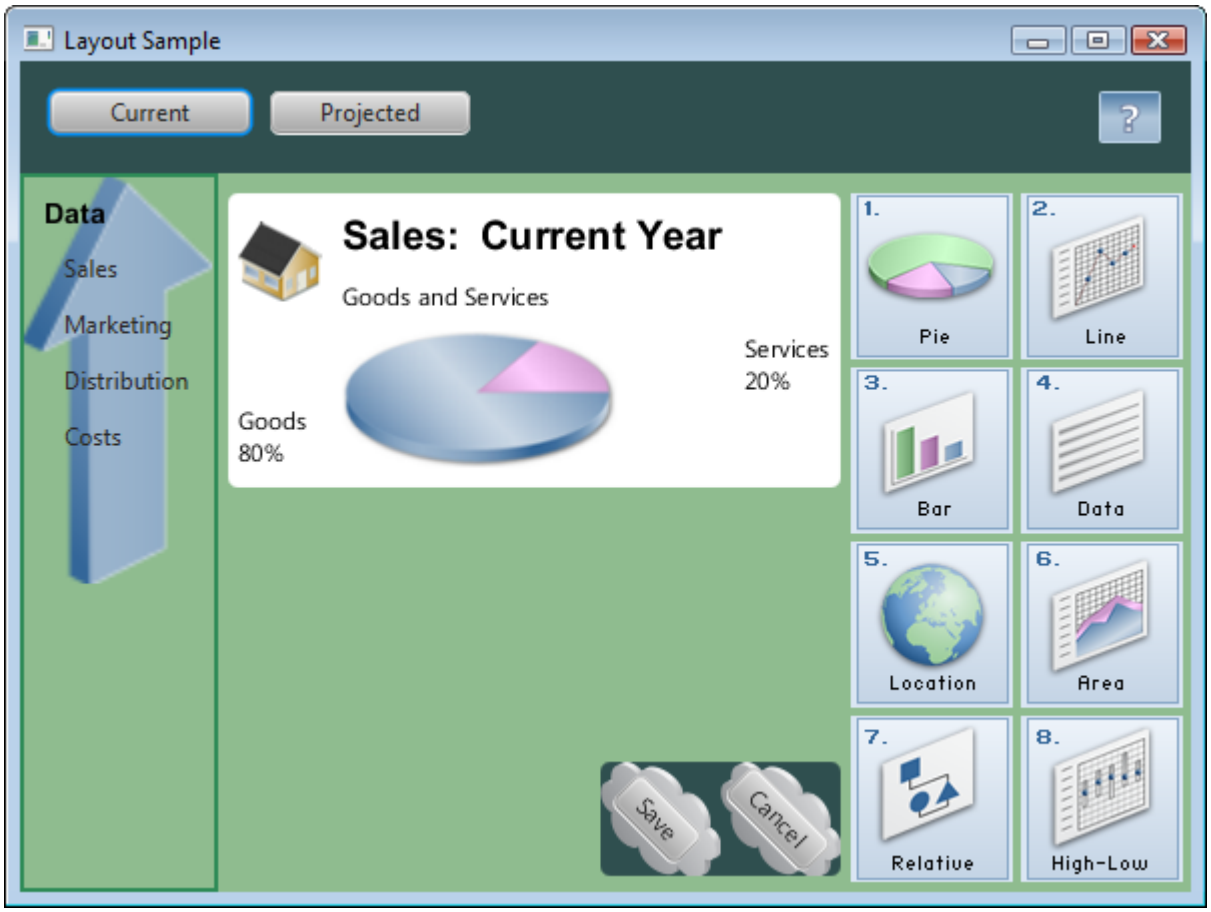
Example 3–14 Assigning a Style to the Anchor Pane

```
AnchorPane anchorpane = new AnchorPane();
anchorpane.getStyleClass().add("pane");
```

Using a Background Image

Images can be used as the background for a layout pane by setting the background image properties. You can specify the image, size, position, and repetition in a style class. Figure 3–3 shows another version of the Layout Sample user interface that uses background images for the `VBox` pane on the left and the `HBox` pane that contains the Save and Cancel buttons.

Figure 3–3 Styled with Images



Example 3–15 shows the style class definitions for adding the background images.

Example 3–15 Styles Using Background Images

```
.vbox {
    -fx-background-image: url("graphics/arrow_t_up_right.png");
    -fx-background-size: 96, 205;
    -fx-background-repeat: no-repeat;
    -fx-border-color: #2e8b57;
    -fx-border-width: 2px;
    -fx-padding: 10;
    -fx-spacing: 8;
```

```
}

#hbox-custom {
    -fx-background-image: url("graphics/cloud.png");
    -fx-background-size: 60,69;
    -fx-padding: 18 3 18 6;
    -fx-background-radius: 5.0;
}

#button-custom {
    -fx-rotate: 45;
    -fx-text-alignment: center;
}
```

Note the following:

- The images are located in the /graphics directory, which is at the same level as the class file for the application.
- The arrow image was smaller than desired and the cloud image was larger than desired, so the images were resized using the `-fx-background-size` property.
- To prevent the arrow from repeating in the background of the `VBox` pane, the `-fx-background-repeat` property is set to `no-repeat`.
- To angle the buttons, a new style class named `#button-custom` is defined, and the ID for the Save and Cancel buttons is set to `button-custom`.

Additional Resources

For more information on CSS and JavaFX, see the following documents:

- JavaFX CSS Reference Guide
- Skinning JavaFX Applications with CSS